수학의 한계와 싸우는 방법들

이장건

September 20, 2022

Abstract

어떠한 문제를 풀기 위해 프로그램을 만든다는 것은 그 문제의 해답을 내놓는 알고리즘을 제공하는것과 같다. 아쉽지만 수학 문제들은 불확정함으로, 모든 문제를 푸는 알고리즘을 내놓을수는 없고, 프로그램이 옳다는걸 증명하는 것 또한 기계적으로 할수 없다. 그렇지만, 이러한 사실에 무너지지 않고 극복하는 것이 사람이다. 정확(exact)한 방법은 없어도, 문제 범위를 제한하는 것과 다양한 휴리스틱(heuristic)을 통해 방대한 발전을 이룰수 있음을 공학자들은 증명해냈고, 앞으로도 계속 문제를 풀어낼 것이다.

매우 슬프지만, 수학은 한계가 명확하다. 완전(complete)하지 못하고, 스스로의 일관성(consisteny)를 증명하지 못하며, 어떠한 문제를 풀수 있는지조차 결정할 수 없다(undecidable). 따라서, 어떠한 문제를 푸는 프로그램을 만드는 것과, 그 프로그램이 정말 문제를 풀었는지를 증명하는 것 또한 수학이기에 완전할수 없다. 그렇지만, 사람은 오히려 이러한 제한을 기쁜 도전으로 받으들이고 다양한 문제를 풀수 있는 프로그램을 개발했으며, 이러한 프로그램들의 증명하는 방법들 또한 계속 발전시켜 나갔다.

모든 문제를 풀수 있는 프로그램을 만들수는 없다. 프로그램이 쓰여진 언어는 튜링 머신의 구현이고, 튜링 머신으로는 결정 불가능한 문제들이 존재한다. 그렇지만 모든 문제를 한번에 푸는 것을 포기하고, 특정 문제만 푸는 것에 집중하면 놀라운 기술들을 만들수 있다. Z3[3]를 보라. 비선형 방정식들의 만족성(satisfiablity)라는 잘 알려는 결정할수 없는 문제를 푸는 과제에 도전했고, 몇년간의 노력과 다양한 휴리스틱을 통한 성능 향상을 거친 후, 지금에는 여러 도구의 핵심적인 백엔드(backend)로서 기능하고 있다. 도구가 아닌 어디에나 쓸수있는 알고리즘만 보더라도, 이동하는 판매원 문제 (traveling salesman problem) 같은 많은 문제가 진화에서 보이는 생물의 발전 방식을 기반으로 풀수 있다는 직관에 기반해, 유전 알고리즘이라는 패러다임이 등장했고 발전했다. 특히, 사람이 보는 모든 문제는 뇌가 푼다는 직관에 의거해, 심층학습이라는 혁신적인 알고리즘을 만들었다. 심층학습, 특히 강화학습의 등장 이후, 여태까지 풀 염두조차 없던 많은 문제가 풀리고 있다. 이처럼 설령 모든 문제를 풀수 없다는것을 알아도, 특정 문제에만 집중하면 사람은 그를 푸는 대단한 방법들을 만들어 왔고, 앞으로도 만들 것을 믿어 의심치 않는다.

모든 프로그램을 단번에 증명하는 방법은 존재하지 않는다. 이는 증명하는 것 또한 수학 문제를 푸는 것이기에 나오는 당연한 한계이다. 어떠한 문제를 푸는 프로그램을 만드는 것과 비교해서, 증명은 *무엇을 증명하는가*를 결정하는 어려운 문제가 있다. 만약 증명하는 언어가 Python이나 Java같은 잘 짜여진 고수준 언어가 아닌, JavaScript 같은 언어의 의미 구조(semantics)가 매우 복잡하거나, C/C++, LLVM 같이 의미 구조, 특히 메모리 관련 의미 구조가 명확하지 않은 대상인 경우 특히 복잡해진다. 어떠한 기반 위에서 증명해야 하는지 불명확한데, 무엇을 증명한단 말인가? 이를 해결하기 위해선, 이러한 의미 구조의 의미를 엄밀하게 먼저 잡는 것이 중요하다고 생각한다. 복잡한 의미 구조에 모순이 없다는걸 보이기 위해, 이러한 일은 iRC11[2]나 Promosing Semantics[4], 또는 상업적으로 사용되는 검증된 C 컴파일러인 CompCert [5]와 같이 Coq이나 다른 엄밀한 증명 보조 도구로 검증되는것이 바람직하다.

아쉽지만, 이러한 증명 보조 도구를 사용하는 것은 증명 방법들이 자명하지 않기에 난이도가 높고, 또한 큰 스케일의 프로그램을 검증하기에는 매우 어렵다. 따라서, Viper [6]나 CBMC [1] 같은 대부분의 검증 도구는 엄밀하지 않은 기반을 두는 대신, 검증의 자동화 비율을 높인다. 검증에서 이러한 일장일단이 있는 것은 바람직하지 않으며, 자동화와 엄밀함을 동시에 가진 도구가 있다면 양쪽의 장점을 모두 가져갈 것이다. 최근, 이러한 도구가 실제로 등장했다! RefinedC [7]는 두 마리의 토끼를 잡는 문제를 혁신적인 타입 시스템을 만듬으로서 해결했다. 비록 다른 자동화 도구에 비해서 사용자가 다루기에는 난이도가 높지만, 언젠가는 이러한 도구가 계속 발전해서 자동화와 엄밀함을 동시에 가진 도구가 나올 것이라고 믿는다.

모든 문제를 한번에 풀수 있는 방법은 존재하지 않는다. 정말 아쉽게도, 모든 문제를 푸는 알고리즘은 존재할수 없고, 모든 정리를 증명하는 방법은 없다. 그러나 이는 어찌 보면 너무 큰 문제를 한번에 풀려고 하는 욕심에서 나오는 당연한 한계이다. 역사가 보여줬듯이, 오히려 이 한계를 통해서 풀려고 하는 문제를 제한해서

보고, 제한해서 봤기에 문제의 핵심을 계속해서 해결하는 식으로 학문은 발전했고, 앞으로도 발전할 것이다. 이 수업에서 앞으로 우리는 다양한 프로그램을 보며, 프로그램이 명세를 만족한다는걸 증명하는 방법을 배우고, 명세를 만족하는 프로그램을 자동으로 생성하는 법을 배우는 것이다. 이 수업을 통해서 진정으로 배울것은, "명세를 주고 증명하고 싶은 프로그램"라는 결정할수 없는 문제를 잘 이해하고, 이 문제를 해결하는 방법론을 생각하는 것이 아닐까?

References

[1] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[2] Hoang-Hai Dang, Jacques-Henri Jourdan, Jan-Oliver Kaiser, and Derek Dreyer. Rustbelt meets relaxed memory. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019.

[3] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, page 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Jeehoon Kang, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis, and Derek Dreyer. A promising semantics for relaxed-memory concurrency. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, page 175–189, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, jul 2009.

[6] P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. In B. Jobstmann and K. R. M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 9583 of *LNCS*, pages 41–62. Springer-Verlag, 2016.

[7] Michael Sammler, Rodolphe Lepigre, Robbert Krebbers, Kayvan Memarian, Derek Dreyer, and Deepak Garg. Refinedc: Automating the foundational verification of c code with refined ownership types. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 158–174, New York, NY, USA, 2021. Association for Computing Machinery.